1 Introduction to Cryptographic protocols

2 State spaces in security protocol analysis

3 Scyther: A short introduction
1. Introduction to Cryptographic protocols

2. State spaces in security protocol analysis

3. Scyther: A short introduction
What is a Cryptographic protocol?

- **Cryptographic protocol**: A formal definition of actions (computations) and message exchanges (communications) between some entities, in order to achieve some claimed security properties.

- Example of claimed security properties:
  - entity authentication
  - key agreement
  - aliveness, etc.

- Crypto protocols usually combine other cryptographic primitives (e.g. encryption schemes, signature schemes etc).
Example: The Needham-Schroeder (N-S) protocols

Two protocols by N-S.

- The Symmetric-Key N-S Protocol:
  - Entities: Two users (Alice, Bob) and a trusted authentication server Auth.
  - Uses symmetric keys shared between each user and Auth ($K_a, K_b$).
  - Protocol Goal (claimed security property): establish a fresh session key between two parties ($K_{ab}$) over an insecure network. The session key is secret from all others.

- The Public-Key N-S Protocol:
  - Entities: Two users (Alice, Bob) and a trusted authentication server (Auth).
  - Uses a public-key pair for each entity $((PK_a, SK_a), (PK_b, SK_b), (PK_{Auth}, SK_{Auth}))$.
  - Protocol Goal (claimed security property): establish a secret session key between two parties ($K_{ab}$) over an insecure network.
**Figure:** Graphical example of Needham Schroeder protocol
What is Wrong with N-S?

- Crypto protocols are error-prone. Crypto protocols require **formal security analysis**.

- The N-S protocol was found **flawed** using an automatic tool (Casper/FDR) 17 years later!

- Vulnerable to **replay attack**: Attacker uses older, compromised value for $K_{ab}$ and then replays $\{K_{ab}, A\}_{K_b}$ to Bob, who is unable to tell that the key is not fresh.

- Flaw was **not detected in the original proof** due to **different assumptions** on the intruder model.
Automatic tools based on formal analysis have been presented in the literature.

Main problem: Security in cryptographic protocols is undecidable.

Tools address undecidability in different ways:
- By restricting the protocol behaviors explored using roles.
- By using abstraction methods.
1. Introduction to Cryptographic protocols

2. State spaces in security protocol analysis

3. Scyther: A short introduction
Symbolic analysis

Symbolic analysis models:
- Possible behavior of legitimate agents executing a security protocol.
- Possible behavior of active intruders.

A security protocol is defined as a finite set of communicating processes, referred to as "roles".
- Roles names can be Client/Server or Initiator/Responder.
- Protocols define exchange of message terms between roles.

Figure: Roles and Terms in N-S example
Symbolic analysis

- A protocol specifies the **behavior** of a number of roles.
  - A mapping from **role names** to **processes**.
  - A role consists of a sequence of **send and receive events**.
- **Process P** defines a **possibly infinite** number of behaviors.
- Each **behavior** is represented as sequences of **events**.
- A **sequence of events** is referred to as a **trace** of the system.
- All behaviors of a process P denoted by set of traces $tr(P)$.

**Figure:** Roles and Terms in N-S example
Roles are executed by Agents.

Each role can be executed any number of times by unbounded number of agents.

Protocol $Q$ with $|\text{dom}(Q)| = n$ roles, $\text{dom}(Q) = r_1, r_2, \ldots, r_n$

$Q(r)(a_1, \ldots, a_n)$ the process that is the instantiation of the role $r$, where $r_1$ is substituted by $a_1$ etc.

For any protocol $Q$, the behavior of the agents is defined by the process:

$$\parallel x \in \text{dom}(Q)!Q(x)(\_, \ldots, \_\,)$$
Symbolic analysis

- Notation 
  - interpreted as the unspecified choice.
  - \( Q(Resp)(\_,\_ \) denotes a single execution of the responder role, with any choice for the agent names.

- \( X \parallel Y \) denotes the process consisting of the parallel composition of the process \( X \) and \( Y \).

- \( !X \) denotes the replication of the process \( X \), i.e. \( !X = X \parallel (!X) \).
Verifying security properties of protocols ≈ checking whether all possible behaviors satisfy desired security properties.

Given a protocol $Q$, the system describing the behavior of the agents in the context of the intruder is defined as:

$$Sys(Q) = Intruder \ || \ ||_{x \in \text{dom}(Q)}!Q(x)(\_ , \ldots , \_)$$

If there exists an attack on (a trace property of) a protocol $Q$, it is represented in the set of traces of the system $Sys(Q)$.

If no trace in $tr(Sys(Q))$ exhibits an attack, there is no attack on the protocol.
Crypto protocol analysis tools usually apply some restrictions.

- Do not explore all elements from the set $tr(Sys(Q))$.
- Protocols are not actually verified in the full system $Sys$ but rather in a subset of the behaviors.
- Subsets can be defined by using a Scenario.

A Scenario is a multi-set of processes.

- $S$: the set of all possible scenarios
- $S_c$: the subset of concrete scenarios in which no unspecified agents ( _) occur.
State spaces in security protocol analysis

- $Sys(Q)$ contains any number of replications of each role.
- $MaxRuns(Q, m)$ system contains only a finite number of replications of each role.
- Let $Q$ be a protocol and let $m$ be a non-negative integer. Then:

$$MaxRuns(Q, m) = Intruder \sum_{i=1}^{m} (\sum_{x \in dom(Q)} Q(x)(\_ , \_ , \_ ))$$
Using a **single honest agent** $a$ and **single compromised agent** $e$, for a protocol with roles $r_1, r_2$, $tr(MaxRuns(Q, 1))$ is equal to:

\[
(\bigcup_{k \in a,e} tr(Scen(r_1(a, k)))) \cup (\bigcup_{k \in a,e} tr(Scen(r_2(k, a))))
\]

This yields a set of four scenarios.
Practical implications

Two possible results of state space analysis:

1. Finding attacks on a protocol
   - If an attack is found, unexplored parts are of little interest.

2. No attack was found.
   - If no attack is found, then we only have some assurance of the correctness of the protocol.
   - State space choices have great impact on analysis results.
   - Scenarios that do not cover all possibilities may result to erroneous output.
   - Even for two honest agents, the simplest protocols already need 42 concrete scenarios to explore exactly all attacks involving two runs.
Most tools are free and open source. Some examples are:

- **Avispa** (Automated Validation of Internet Security Protocols)
- **ProVerif**
- **Casper/FDR**
- **Scyther**
1. Introduction to Cryptographic protocols
2. State spaces in security protocol analysis
3. Scyther: A short introduction
Scyther tool

- Tool by Cas Cremers for the automatic verification of security protocols.
- Can be found at:
  http://users.ox.ac.uk/~coml0529/scyther/
- Python based. Current version is 1.1.3.
- Available for various platforms (Linux, Windows, Mac OS).
- Installation instruction are included in the downloadable Scyther archives.
Scyther tool

- Verifies protocols with unbounded number of sessions.
- Can characterize protocols, yielding a finite representation of all possible protocol behaviors.
- Not required to provide scenarios for property verification, all possible protocol behaviors are explored by default.
- Core elements in a Scyther input file are protocol definitions.
- Has been used to:
  - **analyse** IKEv1, IKEv2 protocol suites and ISO/IEC 9798 family along with a large amount of Authenticated Key Exchange (AKE) protocols.
  - **find** new multi-protocol **attacks** on many existing protocols.
Scyther manipulates terms.

Atomic terms can be any identifier, usually string of alphanumerical characters.

- Constants
- Freshly generated values: random values, declared inside roles using the fresh declaration.
- Variables: Agents can use variables to store received terms.

Atomic terms can be combined into complex terms.

- \((x,y)\): pair of terms \(x\) and \(y\).
- It is allowed to write n-tuples.
Any term can act as a key for **symmetrical encryption**.

Encryption of *ni* with a term *kir* is written as: \( \{ \text{ni} \} \text{kir} \)
- Unless *kir* explicitly defined as part of asymmetric key pair, this is interpreted as symmetric encryption.

**Symmetric-key infrastructure** predefined: \( k(X, Y) \) denotes long-term symmetric key shared between X and Y.

**Public-key infrastructure (PKI)** is predefined: \( sk(X) \) denotes the long-term private key of X and \( pk(X) \) the corresponding public key.

**Example**: Nonce of the initiator (*ni*) encrypted with initiator public key: \( \{ \text{ni} \} \text{pk}(I) \)
recv and send: Receiving and sending a message, respectively. Each send event will have a corresponding recv event.

Claim events: Used in role specifications to model intended security properties.
- Secret: This claim requires secrecy for a given parameter term.
- SKR: Equivalent to the Secret claim. Additionally mark the parameter term as a session-key. Consequence is that using session-key reveal adversary rule will now reveal the parameter term.
- Alive: Aliveness (of all roles).
- Weakagree: Weak agreement (of all roles).

Example: Claim event models that Ni is meant to be secret.
\[ \text{claim}(I, \text{Secret}, Ni); \]
We will explore two examples (both included in the protocol examples of the Scyther tool):

- The symmetric-key N-S protocol.
- The public-key version of the N-S protocol.